Using Schematik Graphics:

User interface

Graphics windows can be selected, printed, resized, scrolled, etc. just like any other window. A close button appears when the corresponding ªgraphics deviceº is closed in scheme (indicating that no more drawing is possible).

Functional graphics

Currently the functional graphics package is only available in SICP mode. It provides procedures for producing images which are automatically displayed by the read-eval-print loop (REPL). All images are rectangular, with a definite width and height. The width and height are measured in printer's points and must be exact integers. (There are about 72 points to the inch.)

```
(line x0 y0 x1 y1 #!optional width height)
```

Produces an image of the specified dimensions with a single line segment. If the `height` isn't specified, it is assumed to be the same as the `width`. If the `width` is also unspecified, a default value is used. The line segment runs from (`x0`, `y0`) to (`x1`, `y1`) in a coordinate system running from ±1 to 1 in each dimension.

`(ps-image string #!optional width height)`
Produces an image with contents specified by the PostScript `string`. The `width` and `height` are defaulted as for `line`. The PostScript `string` is invoked in the context of the same coordinate system as for `line`.

`(resize-image image #!optional width height)`
Produces an image with a suitably scaled version of the same contents as the specified `image`. The `width` and `height` are defaulted as for `line`.

`(quarter-turn-right image)`
Produces a new image formed by turning the specified `image` clockwise 90°.

`(mirror-image image)`
Produces a new image by reflecting the specified `image` about its vertical axis.

`(invert image)`
Produces a new image by reversing black and white in the specified `image`.

`(overlay image . more-images)`
Produces a new image by combining one or more specified images. It's as though they were on transparencies that were

layered together. The images must be of identical width and height.

`(stack top-image . more-images)`
Produces a new image by adjoining the images vertically, with the first one on top and the last on the bottom. The images must be of the same width.

SICP graphics

In the SICP compatibility package, all the usual ªchipmunk styleº graphics operations will work just as in any other implementation. If `(init-graphics)` is not done explicitly, the first graphics operation will implicitly do it.

MIT Scheme graphics

See Chapter 17 of the *MIT Scheme Reference Manual* for general

information on graphics. See also the following sections on implementation limitations and custom operations. The information specific to creating a Schematik device is as follows:

- The ªgraphics device typeº is `schematik-style-graphics-device-type`
- The additional arguments for `make-graphics-device` are a symbol indicating the unit of measure, which may be `pixels` or `points` (`pixel` or `point` also work) and then two numbers indicating the width and height of the desired drawing area in that unit. For example, to create a 2 inch wide by 3 inch high drawing area (given that there are 72 points to an inch) one would do

```
(make-graphics-device
     schematik-style-graphics-device-type
     'points (* 2 72) (* 3 72))
```

Limitations

± The only drawing modes that approximately work are 0, 3, 7, and 15. 0 draws in the background color, the others draw in the foreground color.

± If you use the undocumented timer-interrupt feature of MIT Scheme for anything else, the best-case outcome is that graphics will be much slower unless buffering is enabled.

± Dotted and/or dashed lines will look funny if the scaling ratio between device and virtual coordinates isn't the same in the x and y directions.

Custom operations

± Performing the `print` operation on a graphics device is identical to selecting the corresponding graphics window in Schematik and doing a Print command.

± The `draw-postscript` operation takes one string argument, which is arbitrary Display PostScript, and executes it in the appropriate context. A space character is automatically appended to the end to delimit the last token of the text; this implies that you can not split one token across two `draw-postscript` operations. If any PostScript error results, it will be reported on the console log and the graphics device will be summarily closed. An error may be reported in scheme, or then again it may not. (Of course, if any further attempt is made to use the closed device, an error will be reported.) Any output from PostScript (e.g. the == operator) will once flushed also appear on the console.

± The foreground color can be set using any of three alternative custom operations: `set-foreground-hsb`, `set-foreground-rgb`, and `set-foreground-gray`. Of these, the first two are fully general (i.e. can specify any color), while

the third is limited to shades of gray including white and black. All use real numbers in the range 0±1 to specify the colors. The first two take three arguments in this range, while only one is needed to specify a shade of gray. See the section below for a description of the three color specification systems. Changing the foreground color does not change any existing graphics, but all new drawing will be in the new color (except in drawing mode 0, i.e. erase).

± The background color can be set in the same three ways as the foreground color, using the operations `set-background-hsb`, `set-background-rgb`, and `set-background-gray`. Changing the background color does not change any existing graphics, but when a graphics-clear is done, the entire drawing area will be filled with the current background color. Additionally, any new drawing done in drawing mode 0 (i.e., erase) will be in the new background color.

Color systems

- The three arguments to the HSB operations specify hue, saturation, and brightness in that order. If the brightness is 0, then the color will be black regardless of the other two coordinates. If the brightness is 1, the color will be maximally bright. If the saturation is 0, then the color will be a shade of gray determined by the brightness, independent of the hue. If the saturation is 1, a pure color will result; if you decrease the saturation towards 0, the color gets diluted with progressively more white light. The hue coordinate should be viewed as circular in nature, with both 0 and 1 being red, 1/3 being green, and 2/3 being blue. Intermediate values are mixtures of the neighboring primaries.
- The three arguments to the RGB operations specify the brightness of the red, green, and blue components of the color,

respectively. If all three are equal, a shade of gray results. In particular, all three being 1 is white, while all three being 0 is black.

± The single argument to the `set-xxx-gray` operations specifies just the brightness, with 1 being white, 0 being black.